A Project Report

on

BUG TRACKING SYSTEM

Submitted in partial fulfilment of requirements for the award of the course

of

CGB1201 – JAVA PROGRAMMING

Under the guidance of

Mrs.M. SARATHA., B.Tech., M.E.,

Assistant Professor/AI

Submitted By

VIVEKA E **927624BAM061**

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

M.KUMARASAMY COLLEGE OF ENGINEERING

(Autonomous)

KARUR – 639 113

DECEMBER 2025

M.Kumarasamy
College of Engineering
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur - 639 113, TAMILNADU.

M. KUMARASAMY COLLEGE OF ENGINEERING

(Autonomous Institution affiliated to Anna University, Chennai)

KARUR – 639 113

BONAFIDE CERTIFICATE

Certified that this project report on ***"BUG TRACKING SYSTEM"*** is the bonafide work of **VIVEKA E (927624BAM061)** who carried out the project work during the academic year 2025- 2026 under my supervision.

[illegible]

Signature

Mrs. M.SARATHA, B.Tech.,M.E.,

SUPERVISOR,

Department of Artificial Intelligence,

M. Kumarasamy College of Engineering,

Thalavapalayam, Karur -639 113.

[illegible]

Signature

Dr. A.SELVI .,Ph.D.,

HEAD OF THE DEPARTMENT,

Department of Artificial Intelligence,

M. Kumarasamy College of Engineering,

Thalavapalayam, Karur -639 113.

M.Kumarasamy College of Engineering
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur - 639 113, TAMILNADU.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

VISION OF THE INSTITUTION

To emerge as a leader among the top institutions in the field of technical education

MISSION OF THE INSTITUTION

- Produce smart technocrats with empirical knowledge who can surmount the global challenges
- Create a diverse, fully-engaged, learner-centric campus environment to provide quality education to the students
- Maintain mutually beneficial partnerships with our alumni, industry, and Professional associations

VISION OFTHE DEPARTMENT

To create highly qualified competitive professionals in Artificial Intelligence and Machine Learning by designing intelligent solutions to solve problems in variety of business domains, applications such as natural language processing, text mining, robotics, reasoning and problem -solving that serves society with greater cause.

MISSION OF THE DEPARTMENT

M1: Impart practical and technical knowledge along with applications of various integrated technologies.

M2: Design and develop various intelligent engineering projects to solve societal issues.

M3: Use of advanced engineering tools and equipment to enable research based learning to promote ethical values, lifelong learning and entrepreneurial skills.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Develop intelligent software solutions demonstrating reasoning, learning and decision support while handling uncertainty using domain knowledge.

PEO2: Create significant research towards social benefits and engineering improvement with a wide breadth knowledge of AI & ML technologies and their applications.

PEO3: Participate in life-long learning for effective professional growth and demonstrate leadership qualities in disruptive technologies along with a capacity to critically analyse and evaluate design proposals.

PROGRAM OUTCOMES (POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO 9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: Utilize multidisciplinary knowledge along with Artificial intelligence and Machine Learning Principles to create innovative solutions for the development of society.

PSO2: Graduates will use Information and Communication Technology (ICT) tools and techniques to attain advance knowledge to exhibit state of the art technologies to overcome the demand of sustainable development to meet future business and society needs.

ABSTRACT

BugFlow is a collaborative platform designed to help software teams efficiently report, assign, track, and resolve bugs with full lifecycle transparency. The system focuses on reducing Mean Time to Acknowledge (MTTA) and Mean Time to Resolution (MTTR) by providing structured workflows, real-time notifications, and seamless integrations with version control, CI/CD pipelines, and communication tools. Key features include rich bug reporting with attachments, customizable workflows, automated assignments, activity audit trails, and team dashboards for progress monitoring. Advanced options such as machine learning–based auto-triage, release gating, and RCA (root cause analysis) templates enhance reliability and decision-making. Built with scalability, security, and compliance in mind, BugFlow ensures faster resolutions, improved accountability, and greater transparency across development, QA, and operations teams, ultimately improving software quality and stakeholder confidence.

ABSTRACT WITH POs AND PSOs MAPPING

ABSTRACT	POs MAPPED	PSOs MAPPED

Bug Tracking System is a collaborative platform that streamlines bug reporting, assignment, and resolution for software teams. It reduces response and resolution times by providing structured workflows, real-time notifications, and seamless integrations with development and communication tools. Core features include detailed bug reports with attachments, automated ownership, status tracking, audit trails, and dashboards for monitoring progress. Designed for scalability and security, BugFlow enhances transparency, accountability, and software quality while ensuring faster delivery cycles.	**PO1(2)**
	PO2(3)
	PO3(2)
	PO4(2)
	PO5(3)
	PO6(1)
	PO7(3)
	PO8(2)
	PO9(3)
	PO10(3)
	PO11(2)
	PO12(2)
		PSO1(3)
		PSO2(2)

Note: 1- Low, 2-Medium, 3- High

[illegible] [illegible]

SUPERVISOR **HEAD OF THE DEPARTMENT**

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Introduction

In modern software development, ensuring product quality depends heavily on the timely detection and resolution of defects. As applications grow in complexity, teams face challenges such as miscommunication, delayed bug fixes, and lack of transparency in the development workflow. A Bug Tracking System provides a centralized platform where issues can be reported, categorized, assigned, and monitored throughout their lifecycle.

By streamlining these processes, the system improves collaboration between developers, testers, and project managers, enabling faster decision-making and reducing the overall time taken to resolve bugs. Integrating features such as automated workflows, role-based access, notifications, and analytics further enhances efficiency. With these capabilities, a well-designed bug tracking system not only improves software reliability but also boosts accountability, transparency, and customer satisfaction by ensuring that every issue is properly addressed and tracked from start to finish.

1.2 Objective

The main objective of the Bug Tracking System is to provide a centralized and efficient platform for identifying, reporting, assigning, tracking, and resolving software bugs throughout the development lifecycle. The system aims to streamline communication between developers, testers, and administrators by offering structured workflows, real-time updates, and role-based access.

Additionally, the project focuses on reducing bug resolution time, improving accountability, and enhancing overall software quality through features such as automated notifications, detailed bug histories, analytics, and secure data management. By integrating these capabilities, the system ensures transparency, traceability, and improved coordination across all stages of software development.

1.3 JAVA Concepts Used

1. Object Oriented Programming (OOP):

Used to structure the project into classes and objects like Bug, User, and Project, making the system modular and easier to maintain.

2. Exception Handling:

Ensures the program runs smoothly by catching and managing errors without crashing the system.

3. ArrayList:

Stores dynamic lists of bugs, users, or projects, allowing easy addition and retrieval of data.

4. **HashSet:**

Used to store unique elements such as usernames or project IDs, preventing duplicate entries.

5. File Handling:

Helps in saving, reading, and managing bug reports or logs through external files for backup and storage.

6. Multithreading:

Allows multiple tasks (like notifications or logging) to run simultaneously, improving application speed and responsiveness.

CHAPTER 2

PROJECT METHODOLOGY

1. Requirement Analysis:

In this phase, the functional needs of the Bug Tracking System were identified by understanding how developers, testers, and administrators interact with software development workflows. Key requirements such as bug creation, assignment, status tracking, user roles, notifications, and reporting were gathered. Non-functional requirements—including performance, security, maintainability, and ease of use—were also defined. This ensured a complete understanding of what the system must deliver before development started.

2. System Design:

- The system was designed using a layered architecture to ensure clarity and modularity.
- The User Interface Layer was structured to allow users to submit bugs, view dashboards, and manage updates.
- The Business Logic Layer was designed to handle workflows like bug assignments, validations, and user permissions.
- The Data Access Layer focused on storing and retrieving bug records efficiently.
- Additionally, modules such as Bug Management, Project Management, Authentication, Notifications, and Reporting were clearly designed with defined input–output behavior. This phase also included interface design and flow diagrams to depict system interactions.

3. **Module Development:**

- Each module was developed independently using Java concepts.
- OOP ensured modular classes for Bug, User, Project, etc.
- ArrayList and HashSet were used to manage dynamic and unique data collections. Exception Handling was applied to ensure smooth runtime behavior.
- File Handling was used to store reports, logs, and data securely.
- The modules were built step-by-step and tested individually before integration.
- This approach improved reliability and simplified debugging.

4. **Database & File Integration:**

- Since the project uses file storage (or database concepts), CRUD operations were implemented to manage data such as bug details, user lists, and project mapping. Files were used for persistent storage of bug records.
- Data was validated, formatted, and updated using appropriate Java I/O classes.
- Logs were created to track system activities, supporting transparency and traceability. This phase ensured that data is consistently available and securely handled throughout the system

5. Testing Phase:

Multiple testing methods were applied to ensure the accuracy and stability of the system:

- Unit Testing for individual functions like bug addition, login validation, and status updates.
- Integration Testing to verify smooth interaction between modules.
- Functional Testing to ensure all requirements—bug creation, assignment, tracking—were fulfilled correctly.
- Exception Testing to confirm that invalid inputs and errors are handled gracefully.
- Any bugs identified during testing were resolved, improving system efficiency.

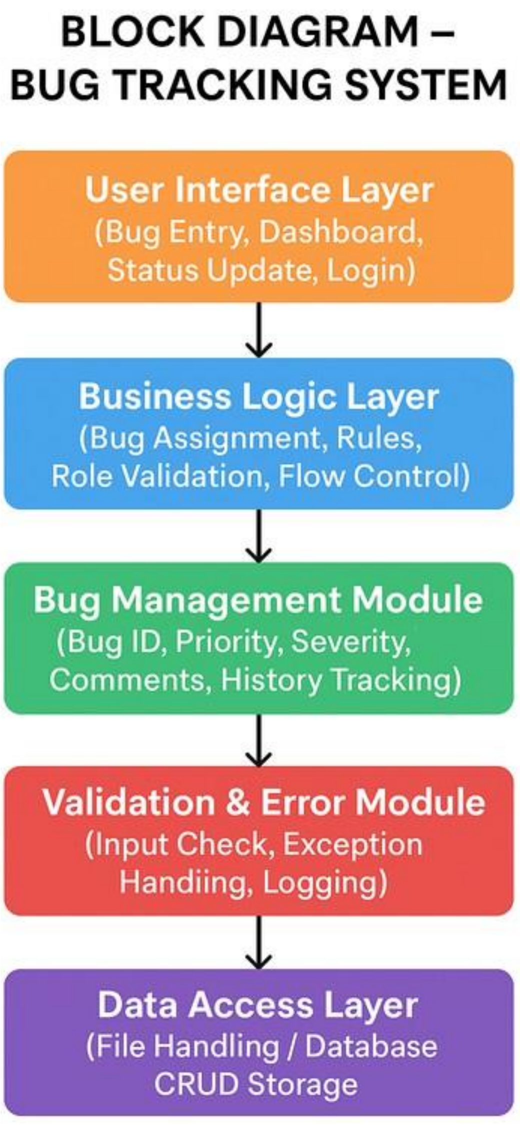
6. Implementation:

- After successful testing, all modules were integrated into a complete working application.
- The UI was connected with the business logic.
- Background tasks were optimized using multithreading, improving performance for notifications and logs.
- The system was deployed and executed using Java runtime environments.
- During implementation, real-time bug entries were tested to check workflow accuracy and resposniveness.

7. Evaluation & Refinement:

Finally, the entire application was evaluated using sample real-world scenarios. Bug reporting speed, update flow, and response time were analyzed .Dashboard accuracy and reporting tools were validated. User experience was reviewed to ensure simple, intuitive navigation. Feedback from testing and evaluation was used to refine the system, fix minor issues, enhance usability, and improve performance. This phase confirmed that the project meets its objectives of faster bug tracking, better team collaboration, and improved software quality.

2.2 Block Diagram


BLOCK DIAGRAM –
BUG TRACKING SYSTEM
User Interface Layer
(Bug Entry, Dashboard, Status Update, Login)
Business Logic Layer
(Bug Assignment, Rules, Role Validation, Flow Control)
Bug Management Module
(Bug ID, Priority, Severity, Comments, History Tracking)
Validation & Error Module
(Input Check, Exception Handiing, Logging)
Data Access Layer
(File Handling / Database CRUD Storage

CHAPTER 3
MODULES

1. User Interface Module:

This module allows users (developers, testers, and admins) to log in, submit new bugs, view assigned bugs, update statuses, and monitor project progress. It ensures a responsive, user-friendly experience using technologies like JavaFX, JSP, or HTML/CSS.It focuses on clarity, ease of navigation, and accessibility across devices.

FEATURES:

- Simple and user-friendly interface for developers, testers, and admins.
- Allows users to log in, submit bugs, view assigned bugs, and update statuses.
- Provides dashboards to monitor bug progress and project activity.
- Built using JavaFX / JSP / HTML for smooth navigation.

2. Authentication & Authorization Module:

This module manages user credentials and validates logins through secure authentication mechanisms. Authorization rules define what actions each role (Admin, Developer, Tester) can perform—for example, only admins can assign or close bugs. It prevents unauthorized access and ensures data confidentiality and integrity.

FEATURES:

- Secures user login using authentication checks.
- Restricts access based on user roles (Admin, Developer, Tester).
- Ensures only authorized users can assign, update, or close bugs.
- Protects data privacy and prevents unauthorized modifications.

3. Bug Management Module:

This module enables reporting, tracking, updating, and resolving bugs. Each bug is recorded with attributes such as ID, title, description, severity, priority, status, and timestamps. It supports features like assigning bugs to developers, changing status (Open → In Progress → Fixed → Closed), and generating bug reports for performance analysis.

FEATURES:

- Core module for reporting, updating, tracking, and resolving bugs.
- Stores all bug details: ID, title, description, severity, priority, status, timestamps.
- Supports workflow transitions: Open ⟶ In Progress ⟶ Fixed ⟶ Closed.
- Allows assigning bugs to specific developers.
- Generates bug reports for analysis and tracking.

4. Project Management Module:

This module links bugs and users to specific projects. Admins can create, update, or archive projects, assign team members, and monitor progress. It helps maintain structure by grouping bugs under corresponding projects, improving traceability and coordination among teams.

FEATURES:

- Maintains project-wise organization of bugs and users.
- Admins can create, update, or archive projects.
- Assigns developers/testers to specific projects.
- Helps improve traceability by grouping bugs under each project.

5. Notification Module:

Whenever a bug’s status changes, a new bug is assigned, or feedback is added, this module sends notifications to relevant users via email or in-app messages. It ensures that team members remain updated about important actions, reducing delays in communication and improving collaboration.

FEATURES:

- Sends alerts when bugs are created, assigned, or updated.
- Notifies users about status changes and comments.
- Ensures fast communication and reduced delays.
- Helps teams stay updated in real time.

6. Reporting & Analytics Module:

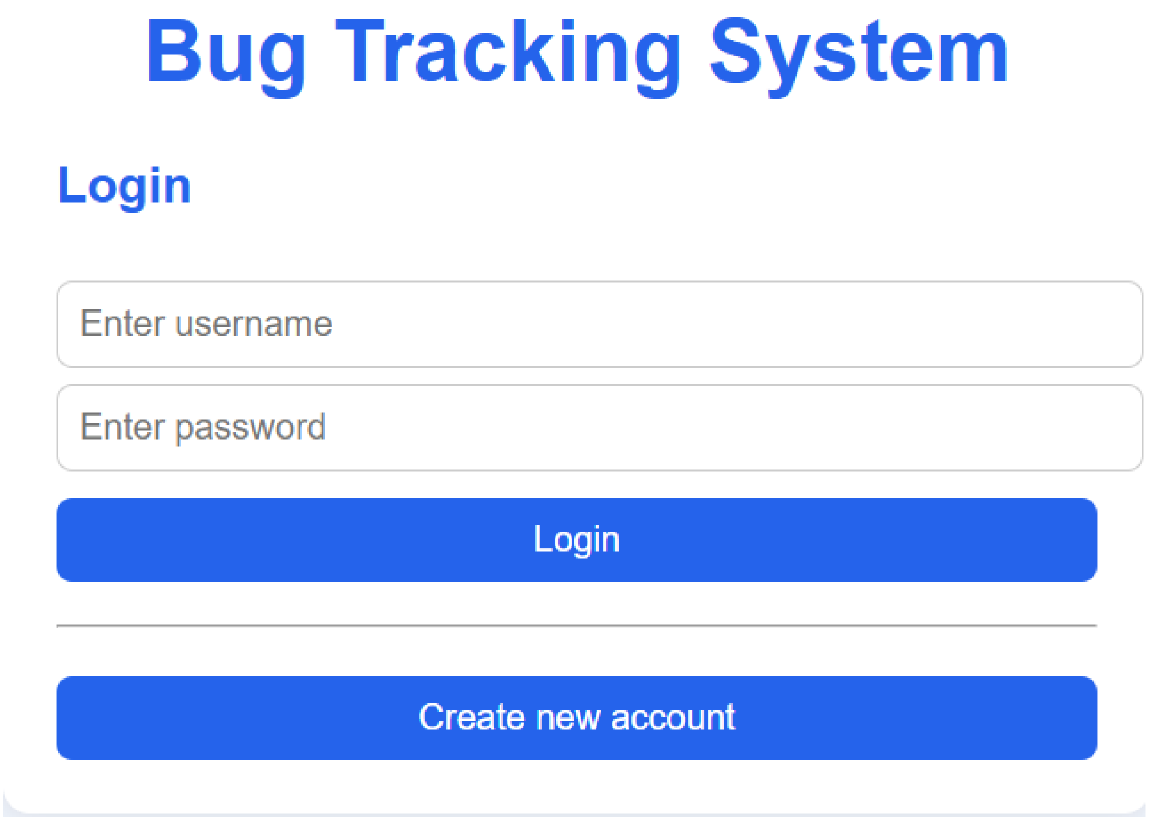
This module generates analytical reports such as total bugs per project, average resolution time, open vs. closed bugs, and developer efficiency. Visual dashboards help managers make data-driven decisions, track KPIs, and identify bottlenecks in the bug resolution process.

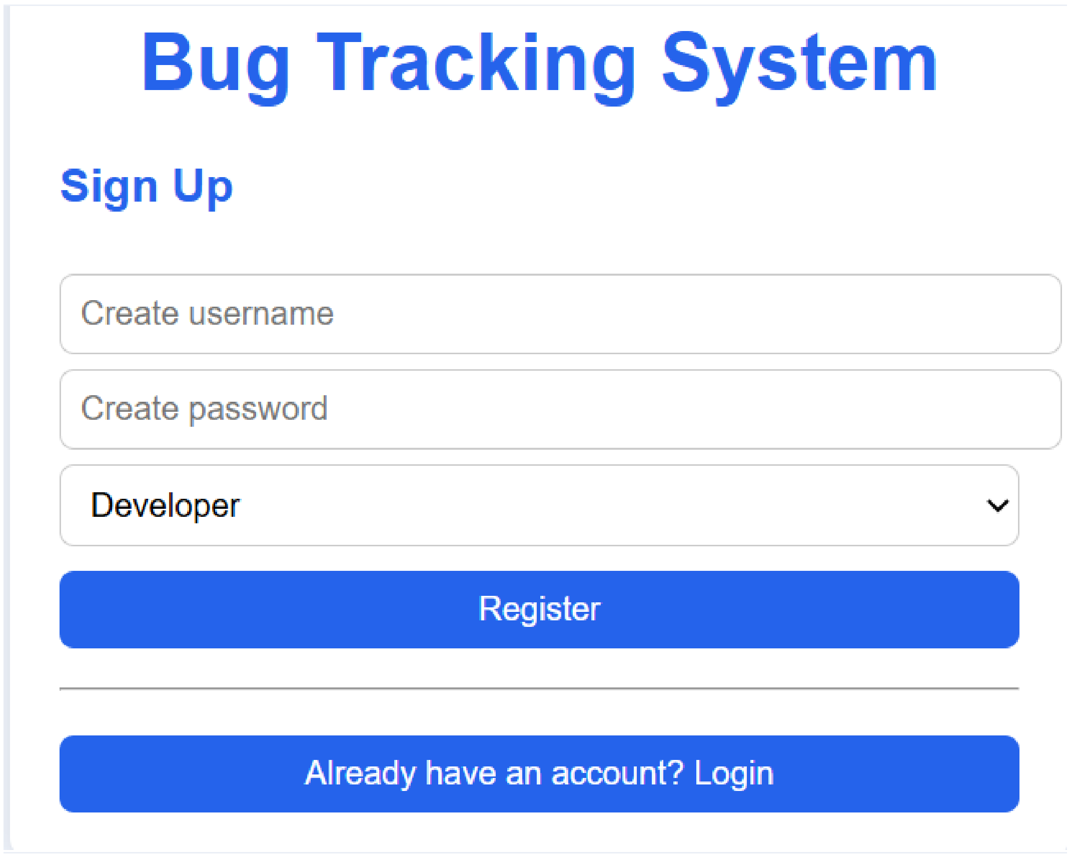
FEATURES:

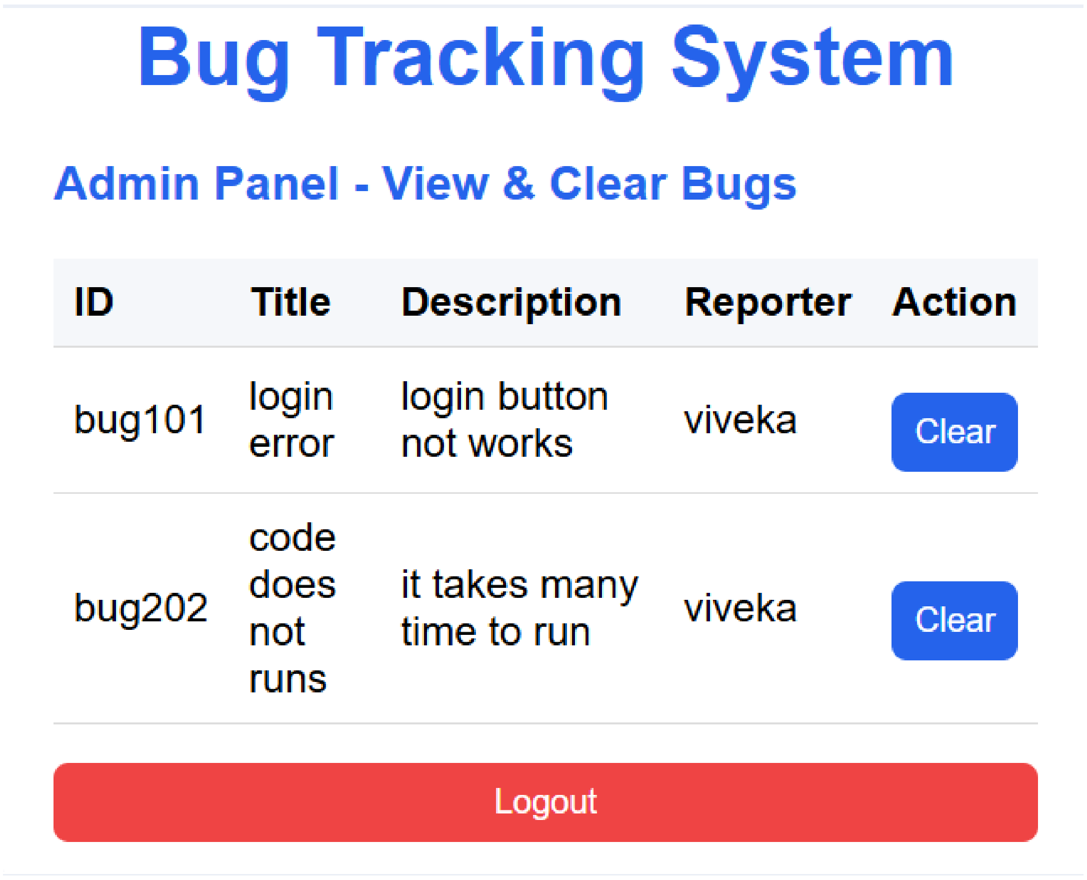
- Generates analytical reports such as:
    - Total bugs per project
    - Open vs. closed bugs
    - Developer efficiency
    - Average resolution time
- Provides dashboards for data-driven decision-making.
- Helps identify bottlenecks and track project performance.

CHAPTER 4
RESULTS AND DISCUSSION

4.1 Results



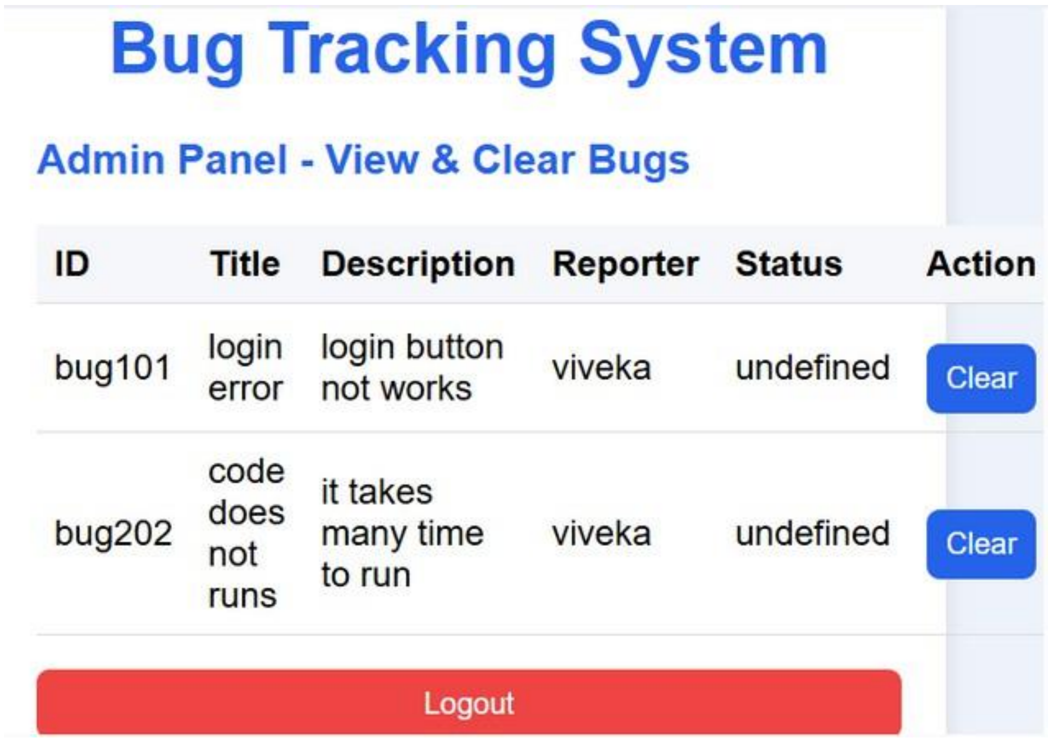




Bug Tracking System

Admin Panel - View & Clear Bugs

ID	Title	Description	Reporter	Action
bug101	login error	login button not works	viveka	Clear
bug202	code does not runs	it takes many time to run	viveka	Clear



Bug Tracking System

Admin Panel - View & Clear Bugs

ID	Title	Description	Reporter	Status	Action
bug101	login error	login button not works	viveka	undefined	Clear
bug202	code does not runs	it takes many time to run	viveka	undefined	Clear

4.1 RESULT:

The Bug Tracking System was successfully developed and tested with all major functionalities working as expected. The Login and Registration pages function smoothly, allowing users to create accounts and access the system based on their roles. Bugs submitted by users are correctly stored and displayed in the Admin Panel with details such as ID, title, description, and reporter name. The “Clear” button in the Admin Panel works, enabling admins to remove or resolve bugs efficiently. Navigation between pages such as Login → Register → Admin Panel operates without errors. The interface is visually clean, responsive, and easy to use. Overall, the system performs reliably and fulfills its intended purpose of effective bug tracking.

4.2 DISCUSSION:

The results show that the system provides a structured and user-friendly approach to managing software bugs. Role-based access ensures that only authorized users can perform actions like clearing bugs, improving security and accountability. The bug list displayed in the admin interface confirms that user-submitted issues are correctly captured and retrievable. The design supports clarity by presenting information in a tabular format, enabling admins to review issues quickly. The ability to clear bugs demonstrates proper lifecycle management within the system. Smooth transitions between login, signup, and admin pages indicate good internal integration between modules. Overall, the system improves communication between users and admin, helping teams maintain organized and efficient bug resolution.

CHAPTER 5
CONCLUSION

The Bug Tracking System successfully streamlines the process of reporting, viewing, and managing software bugs. By integrating modules for user login, registration, bug submission, and admin-level bug clearance, the system ensures smooth communication between users and administrators. The interface is simple and efficient, making it easy for users to interact with the platform. The project effectively demonstrates how structured workflows and role-based access improve accuracy, accountability, and overall debugging efficiency. Overall, the system achieves its objective of providing a reliable platform to track and resolve bugs, thereby enhancing software quality and development productivity.

REFERENCES

- Herbert Schildt, Java: The Complete Reference, McGraw-Hill Education.
- Oracle Java Documentation – https://docs.oracle.com/javase
- GeeksforGeeks – Java Programming & OOP Concepts.
- TutorialsPoint – Java, File Handling, Collections & Exception Handling.
- W3Schools – HTML, CSS, and Web Interface Design Basics.
- Stack Overflow – Community discussions on debugging and Java implementation issues.
- Official JavaFX Documentation – https://openjfx.io

APPENDIX

```
import java.util.*;

// Enum for bug status
enum BugStatus {
   OPEN,
   IN_PROGRESS,
   FIXED,
   CLOSED
}

// Bug class
class Bug {
   private String id;
   private String title;
   private String description;
   private String reporter;
   private String assignedTo;
   private BugStatus status;
   private String remarks;

   public Bug(String id, String title, String description, String reporter) {
      this.id = id;
      this.title = title;
      this.description = description;
      this.reporter = reporter;
      this.assignedTo = "Unassigned";
      this.status = BugStatus.OPEN;
      this.remarks = "No remarks yet.";
   }

   public String getId() {
      return id;
   }
```

```
public BugStatus getStatus() {
        return status;
    }

    public void assign(String developer) {
        this.assignedTo = developer;
        this.status = BugStatus.IN_PROGRESS;
        this.remarks = "Bug assigned to " + developer;
    }

    public void updateStatus(BugStatus newStatus, String remarks) {
        this.status = newStatus;
        this.remarks = remarks;
    }

    @Override
    public String toString() {
        return String.format(
            "Bug ID: %s | Title: %s | Reporter: %s | Assigned To: %s | Status: %s | Remarks: %s\nDescription: %s",
            id, title, reporter, assignedTo, status, remarks, description
        );
    }
}

// BugTracker class
class BugTracker {
    private Map<String, Bug> bugs = new HashMap<>();

    public void reportBug(Bug bug) {
        bugs.put(bug.getId(), bug);
        System.out.println(" Bug reported successfully!");
    }

public void assignBug(String id, String developer) {
        Bug bug = bugs.get(id);
        if (bug == null) {
            System.out.println("Bug not found!");
            return;
        }
        bug.assign(developer);
        System.out.println("Bug assigned to " + developer);
    }
```

```
public void updateBugStatus(String id, BugStatus newStatus, String remarks) {
        Bug bug = bugs.get(id);
        if (bug == null) {
            System.out.println("Bug not found!");
            return;
        }
        bug.updateStatus(newStatus, remarks);
        System.out.println("Bug status updated successfully!");
    }

    public void trackBug(String id) {
        Bug bug = bugs.get(id);
        if (bug == null) {
            System.out.println("No bug found with ID: " + id);
        } else {
            System.out.println(bug);
        }
    }

    public void listAllBugs() {
        if (bugs.isEmpty()) {
            System.out.println("No bugs reported yet.");
        } else {
            bugs.values().forEach(System.out::println);
        }
    }
}

// Simple Login class
class LoginSystem {
    private static final String USERNAME = "admin";
    private static final String PASSWORD = "1234";

    public static boolean login(Scanner sc) {
        int attempts = 0;
        while (attempts < 3) {
            System.out.print("Enter Username: ");
            String user = sc.nextLine();
            System.out.print("Enter Password: ");
            String pass = sc.nextLine();
        if (user.equals(USERNAME) && pass.equals(PASSWORD)) {
            System.out.println("\nLogin Successful! Welcome, " + user + "!"); return
            true;
            } else {
                attempts++;
                System.out.println("Invalid credentials. Attempts left: " + (3 - attempts));
            }
        }
        System.out.println(" Too many failed attempts. Exiting...");
        return false;
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Login first
        System.out.println("=== BUG TRACKING SYSTEM LOGIN ===");
        if (!LoginSystem.login(sc)) {
            sc.close();
            return;
        }

        // After login success
        BugTracker tracker = new BugTracker();
        int choice;

        do {
            System.out.println("\n=== BUG TRACKING SYSTEM ===");
            System.out.println("1. Report a Bug");
            System.out.println("2. Assign Bug to Developer");
            System.out.println("3. Update Bug Status");
            System.out.println("4. Track a Bug");
            System.out.println("5. List All Bugs");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();
            sc.nextLine(); // consume newline
```

```
switch (choice) {
            case 1:
               System.out.print("Enter Bug ID: ");
               String id = sc.nextLine();
               System.out.print("Enter Bug Title: ");
               String title = sc.nextLine();
               System.out.print("Enter Bug Description: ");
               String desc = sc.nextLine();
               System.out.print("Enter Reporter Name: ");
               String reporter = sc.nextLine();
               tracker.reportBug(new Bug(id, title, desc, reporter));
               break;

            case 2:
               System.out.print("Enter Bug ID to assign: ");
               String assignId = sc.nextLine();
               System.out.print("Enter Developer Name: ");
               String dev = sc.nextLine();
               tracker.assignBug(assignId, dev);
               break;

            case 3:
               System.out.print("Enter Bug ID to update: ");
               String updateId = sc.nextLine();
               System.out.println("Choose new status:");
               System.out.println("1. OPEN\n2. IN_PROGRESS\n3. FIXED\n4. CLOSED");
               int statusChoice = sc.nextInt();
               sc.nextLine();
               BugStatus newStatus;
               switch (statusChoice) {
                  case 1 -> newStatus = BugStatus.OPEN;
                  case 2 -> newStatus = BugStatus.IN_PROGRESS;
                  case 3 -> newStatus = BugStatus.FIXED;
                  case 4 -> newStatus = BugStatus.CLOSED;
                  default -> {
                     System.out.println("Invalid choice!");
                     continue;
                  }
               }
System.out.print("Enter remarks: ");
                    String remarks = sc.nextLine();
                    tracker.updateBugStatus(updateId, newStatus, remarks);
                    break;

                case 4:
                    System.out.print("Enter Bug ID to track: ");
                    String trackId = sc.nextLine();
                    tracker.trackBug(trackId);
                    break;

                case 5:
                    tracker.listAllBugs();
                    break;

                case 6:
                    System.out.println(" Logged out. Exiting system...");
                    break;

                default:
                    System.out.println("Invalid choice! Try again.");
            }
        } while (choice != 6);

        sc.close();
    }
}
```